CS 591 K1:
Data Stream Processing and Analytics

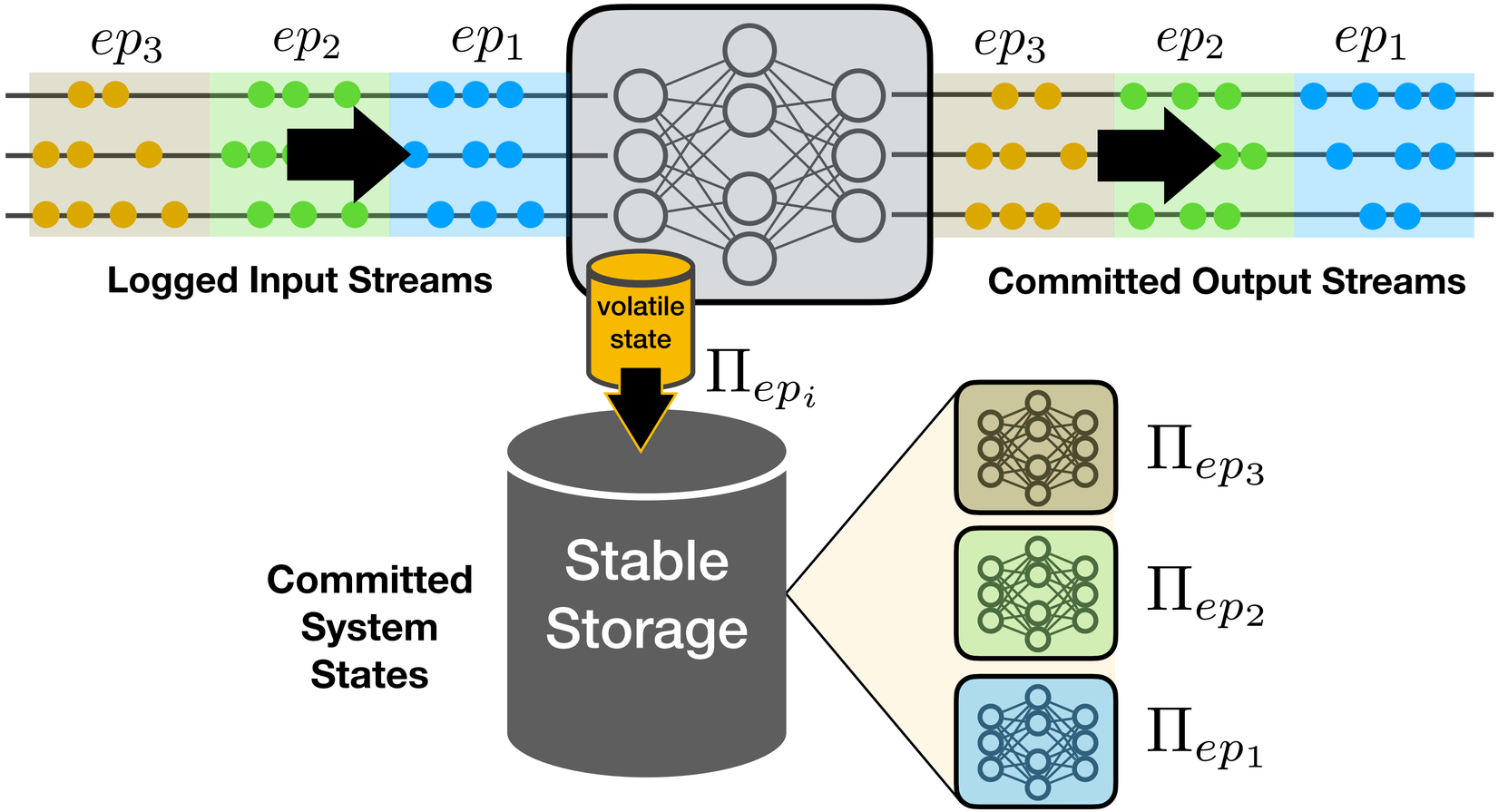
Spring 2021

Exactly-once fault-tolerance in Apache Flink

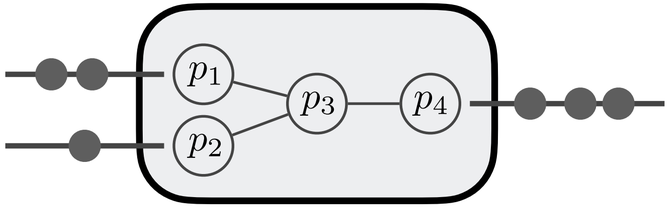
Vasiliki (Vasia) Kalavri
vkalavri@bu.edu

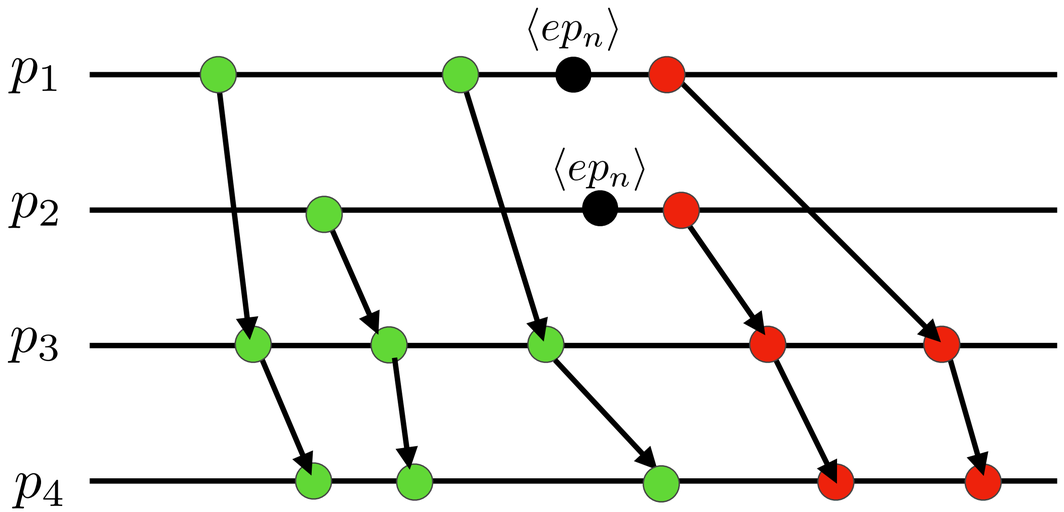
Can we apply this algorithm to retrieve a consistent snapshot of a stream processing application?

Epoch-Based Stream Execution

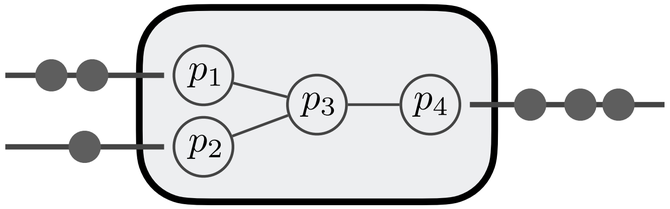


Validity is not sufficient

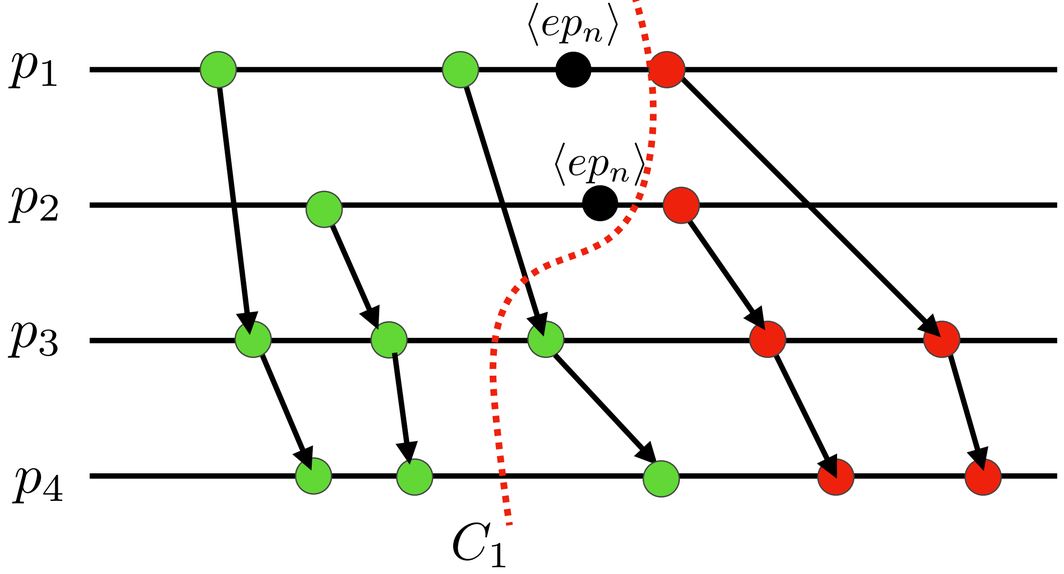




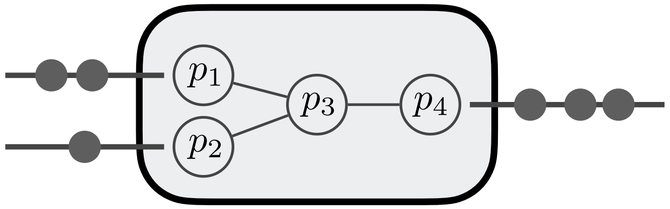
Validity is not sufficient



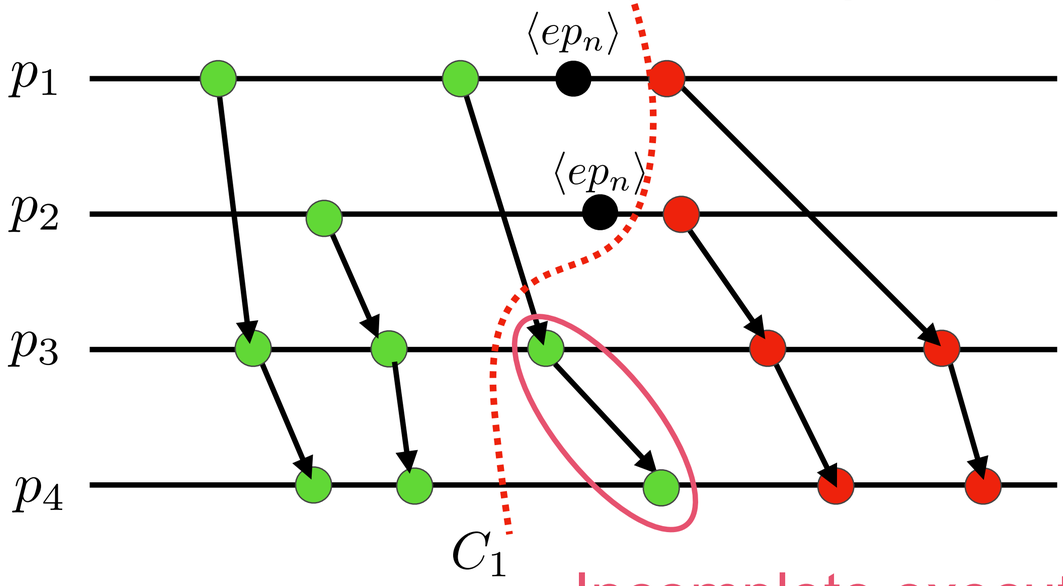
chandy-lamport snapshot



Validity is not sufficient



chandy-lamport snapshot



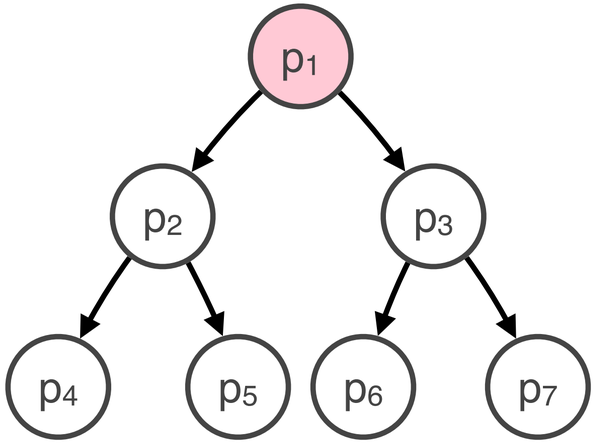
Incomplete execution of an epoch

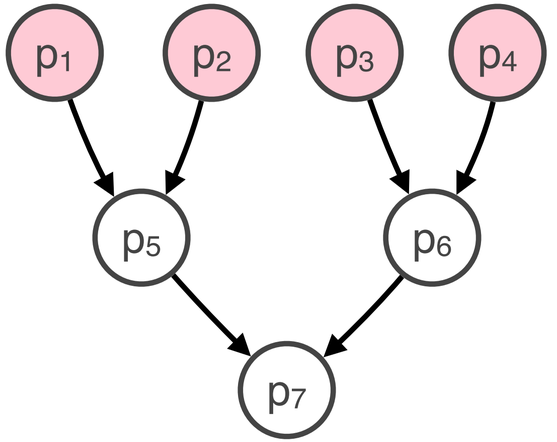
Algorithm Generalizations

The marker-forwarding logic itself guarantees **validity:**

Each local snapshotting action produces markers that **separate** pre-snapshot and post-snapshot events (order maintained by FIFO channels)

Termination is satisfied if initiator can reach all tasks (possible in DAGs via multiple initiators, e.g., sources.)

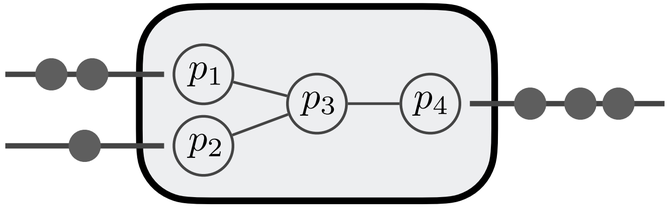


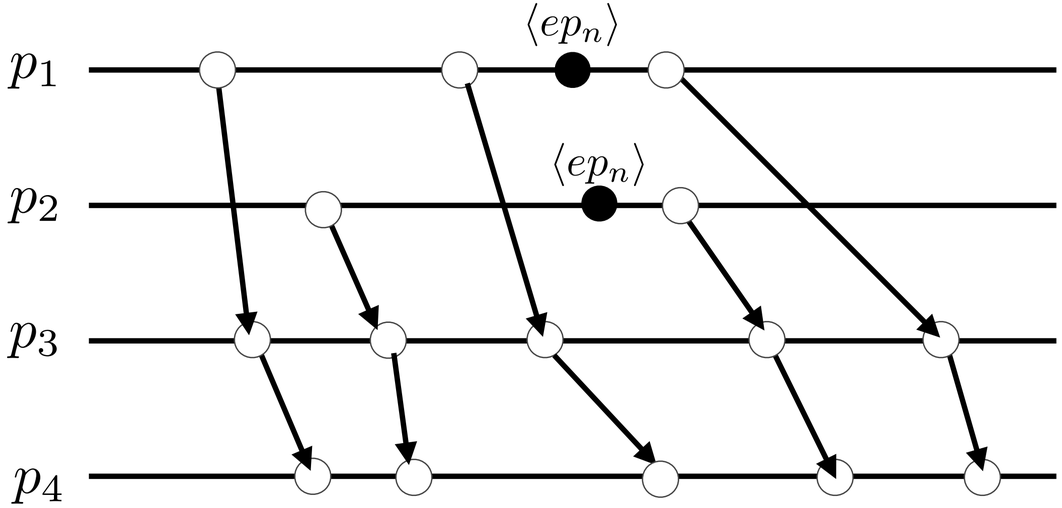


Epoch Snapshotting

- Assumptions:
  - DAG of tasks
  - Epoch change events triggered on each source task (⟨ep1⟩,⟨ep2⟩,…)
    - Issued by a coordinator or generated periodically
- We want to snapshot stream process graphs after the complete computation of an epoch.

Epoch cuts

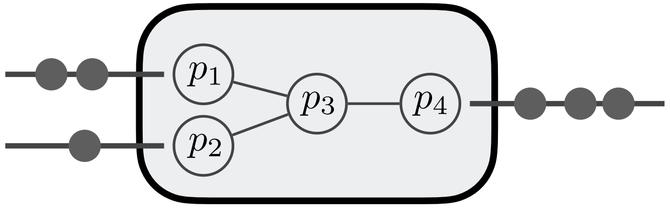


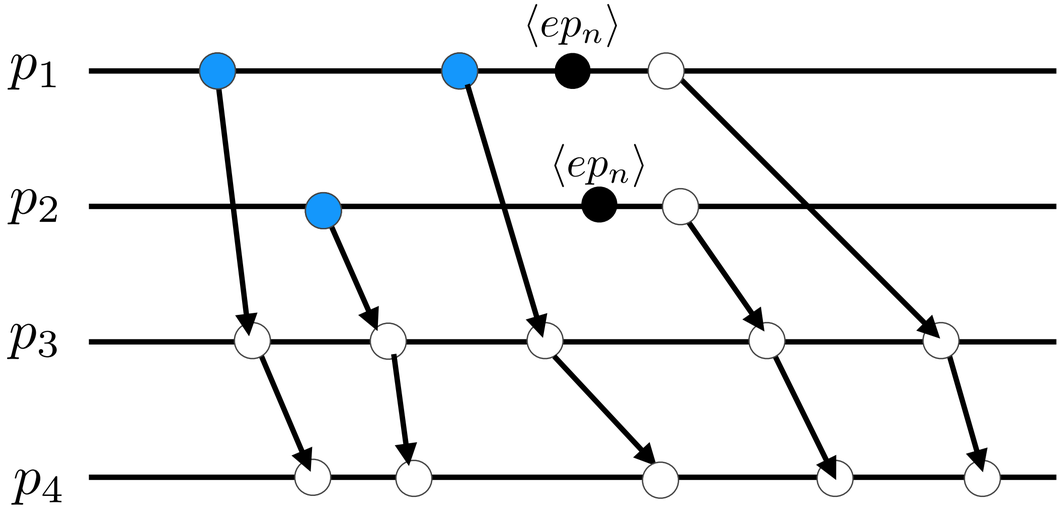


A *epoch-complete* consistent cut that includes events that

Epoch-Completeness: Obtain an epoch-complete system configuration

Epoch cuts



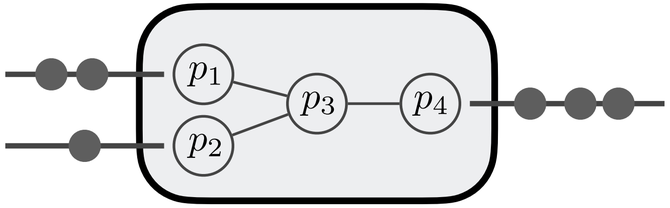


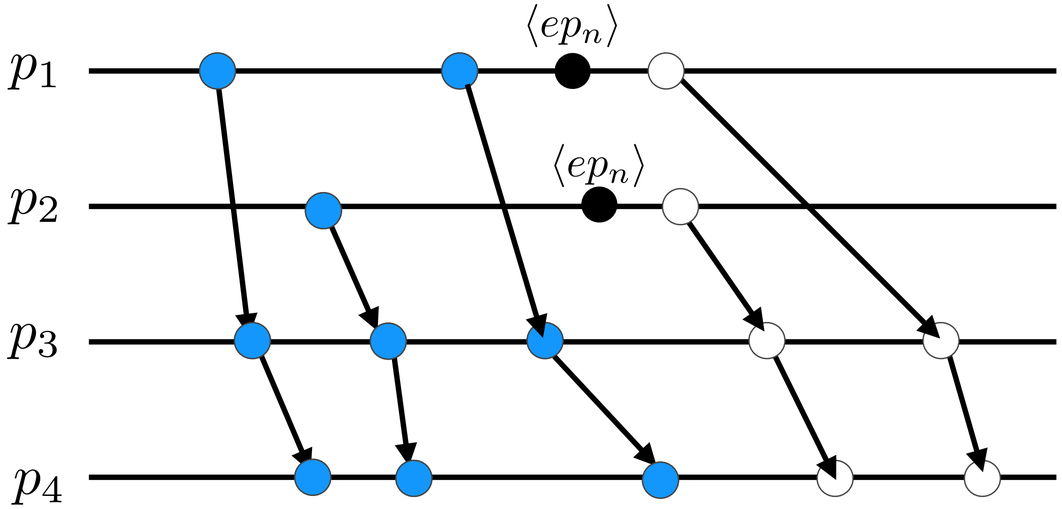
A *epoch-complete* consistent cut that includes events that

1. precede epoch change

Epoch-Completeness: Obtain an epoch-complete system configuration

Epoch cuts





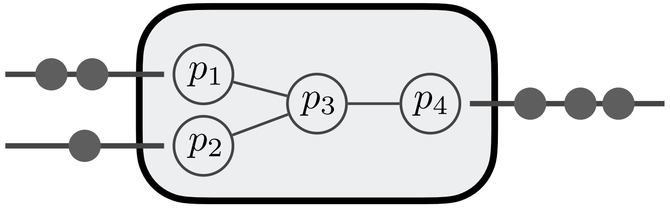
A *epoch-complete* consistent cut that includes events that

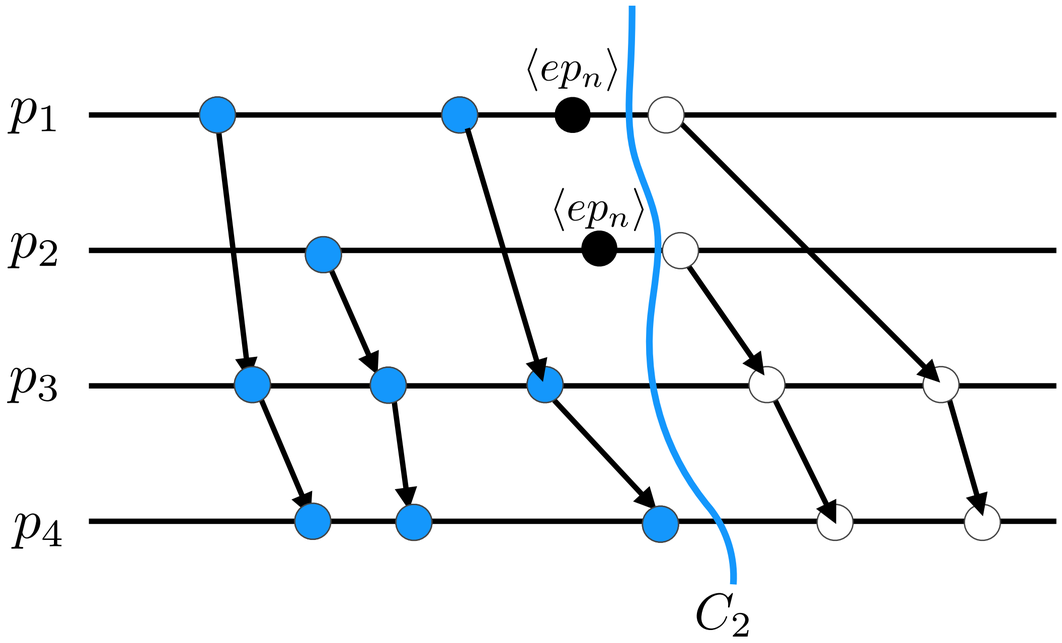
1. precede epoch change

2. are produced by events in cut

Epoch-Completeness: Obtain an epoch-complete system configuration

Epoch cuts





A *epoch-complete* consistent cut that includes events that

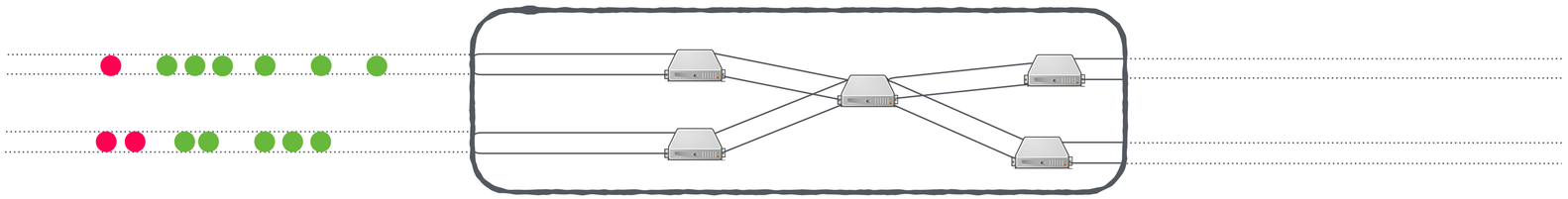
1. precede epoch change

2. are produced by events in cut

3. do **not** causally succeed epoch change

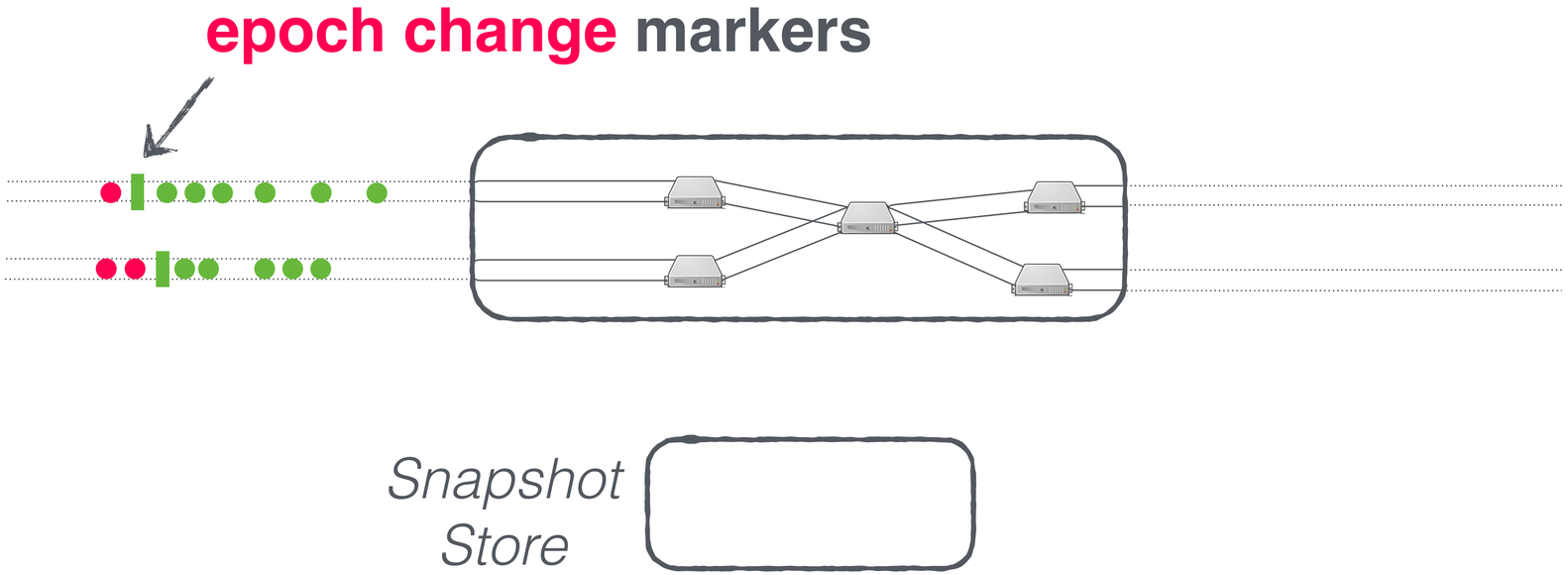
Epoch-Completeness: Obtain an epoch-complete system configuration

Epoch Snapshotting Algorithm

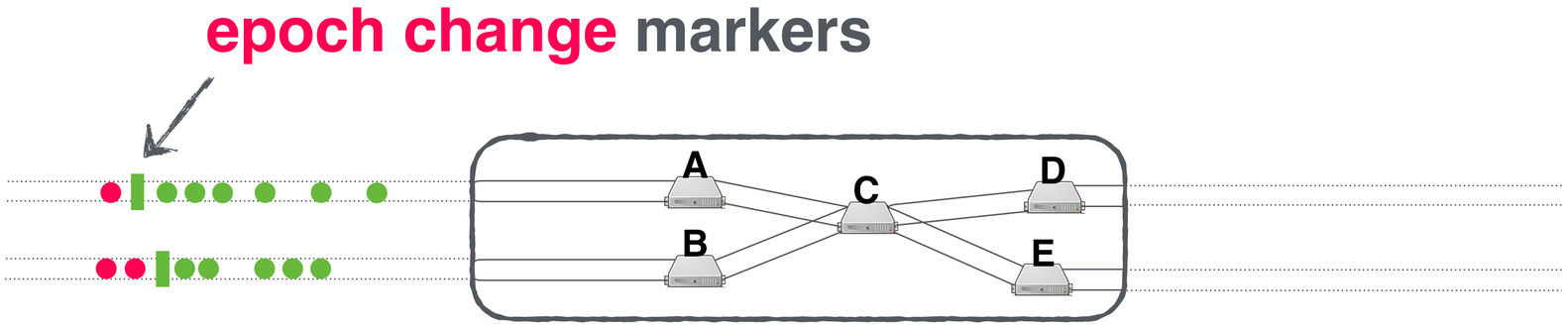


Snapshot Store

Epoch Snapshotting Algorithm

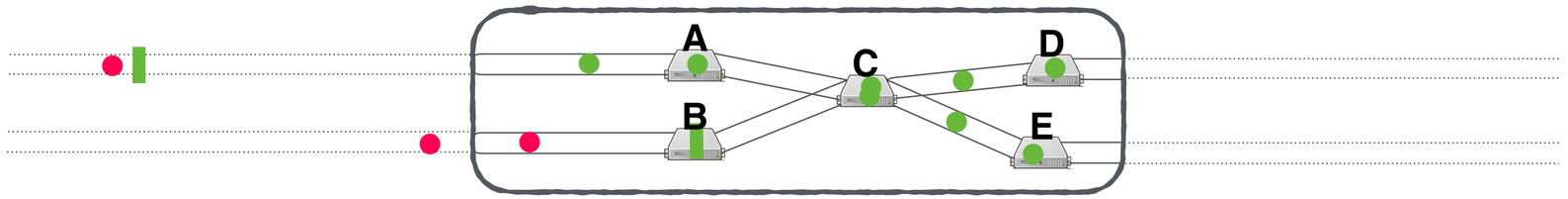


Epoch Snapshotting Algorithm



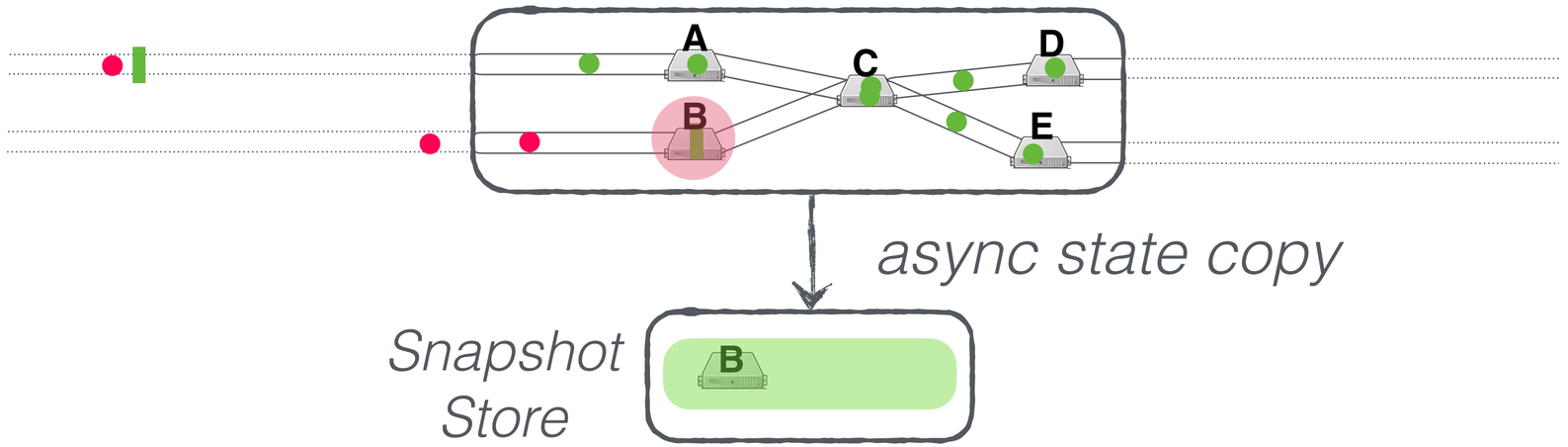
Snapshot Store

Epoch Snapshotting Algorithm

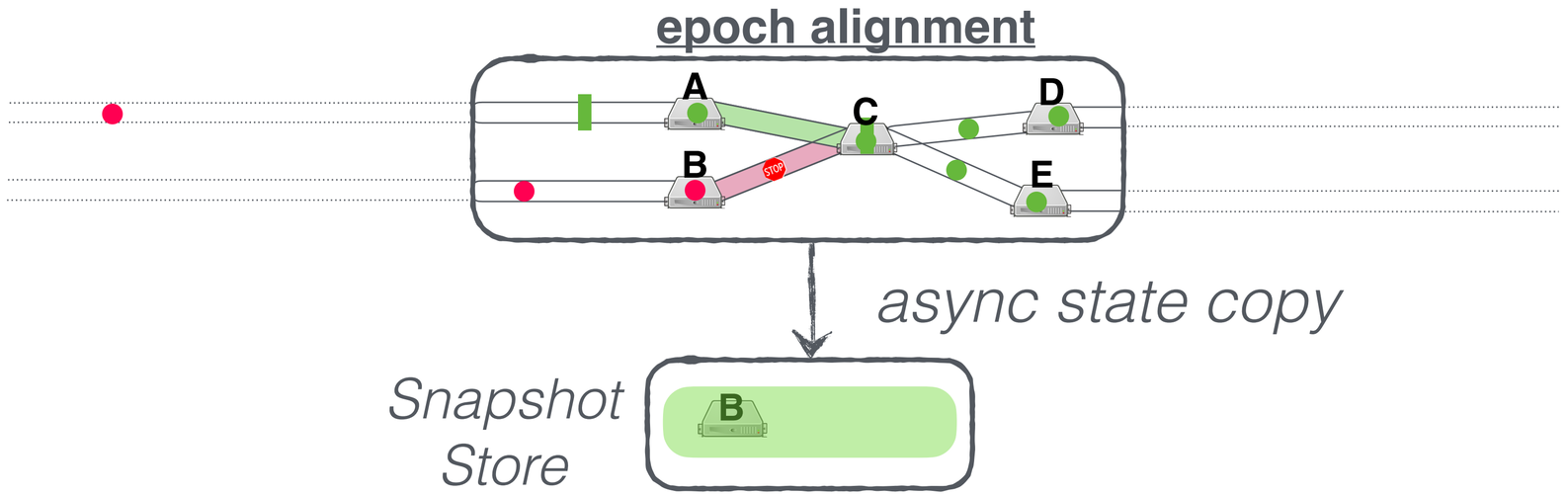


Snapshot Store

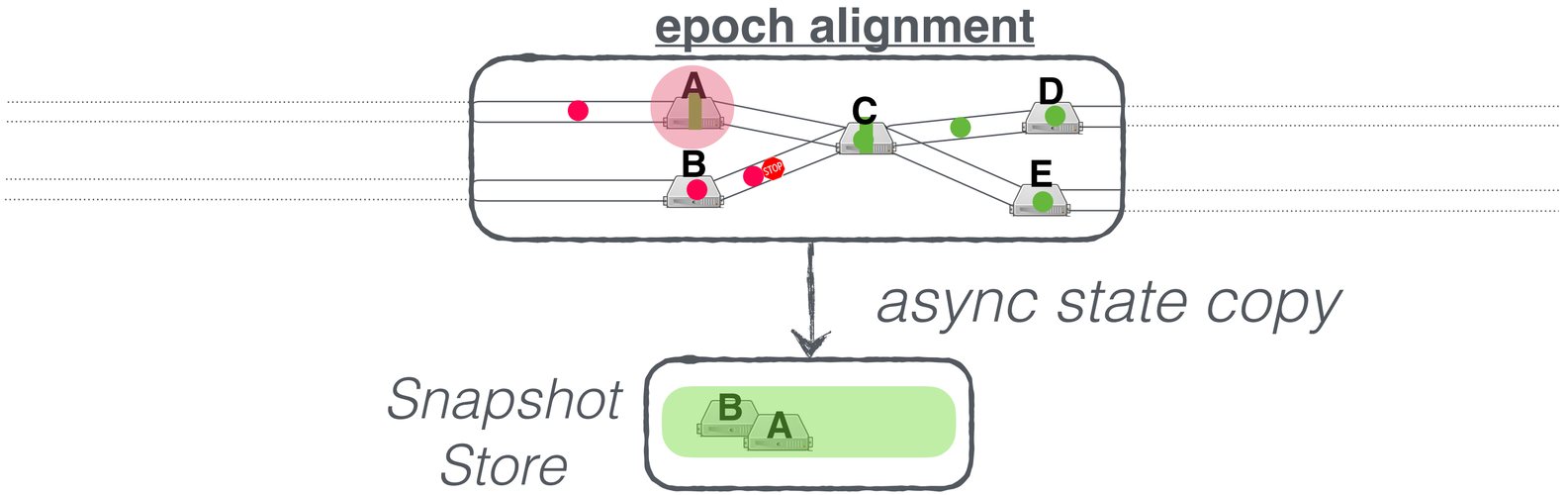
Epoch Snapshotting Algorithm



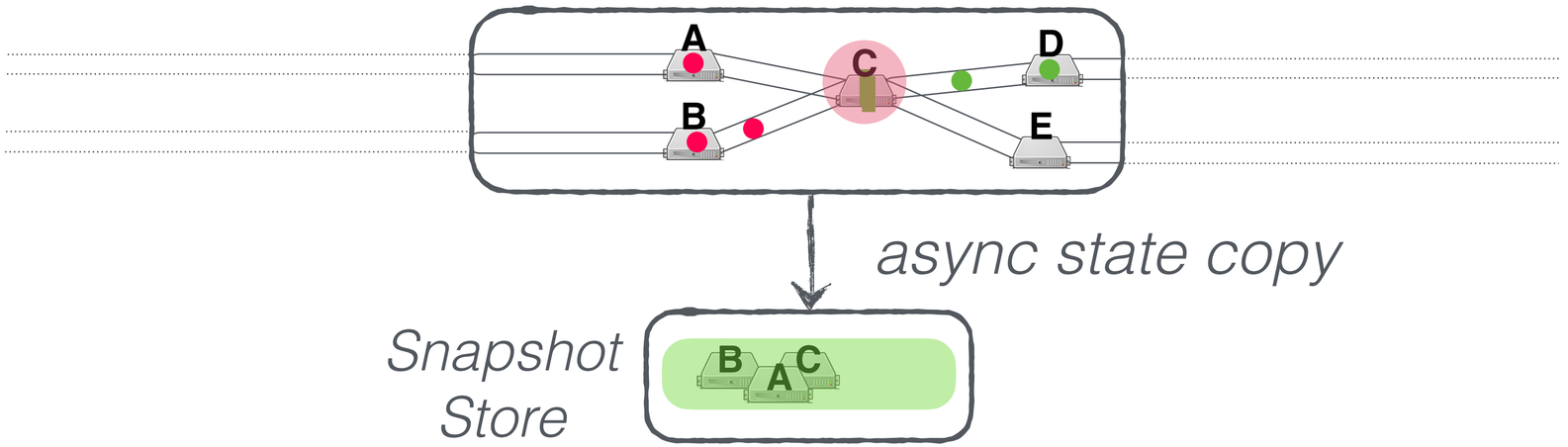
Epoch Snapshotting Algorithm



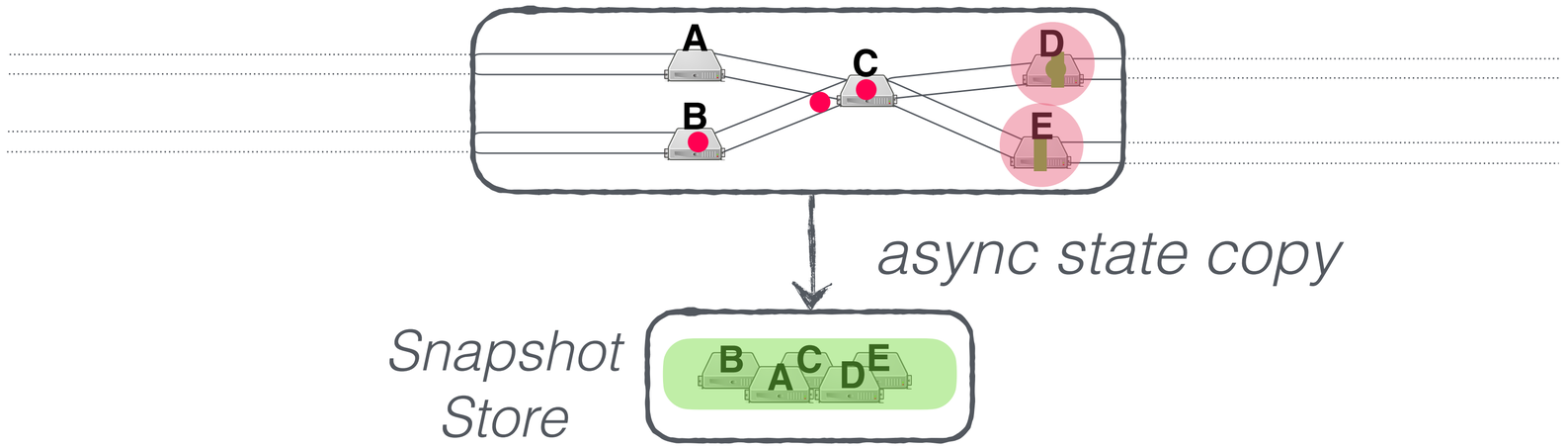
Epoch Snapshotting Algorithm



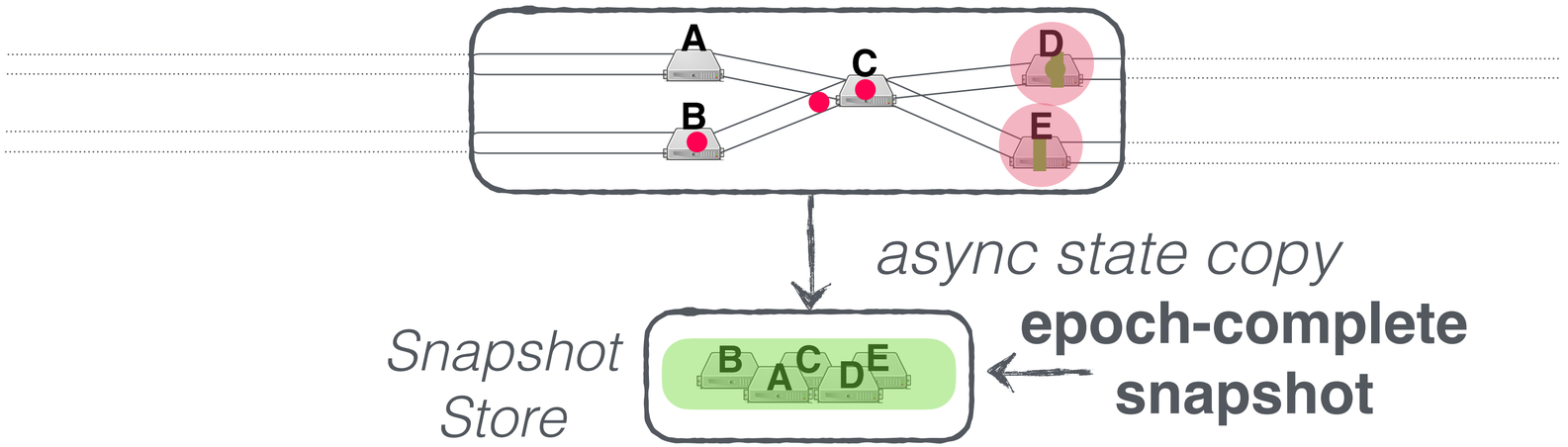
Epoch Snapshotting Algorithm



Epoch Snapshotting Algorithm



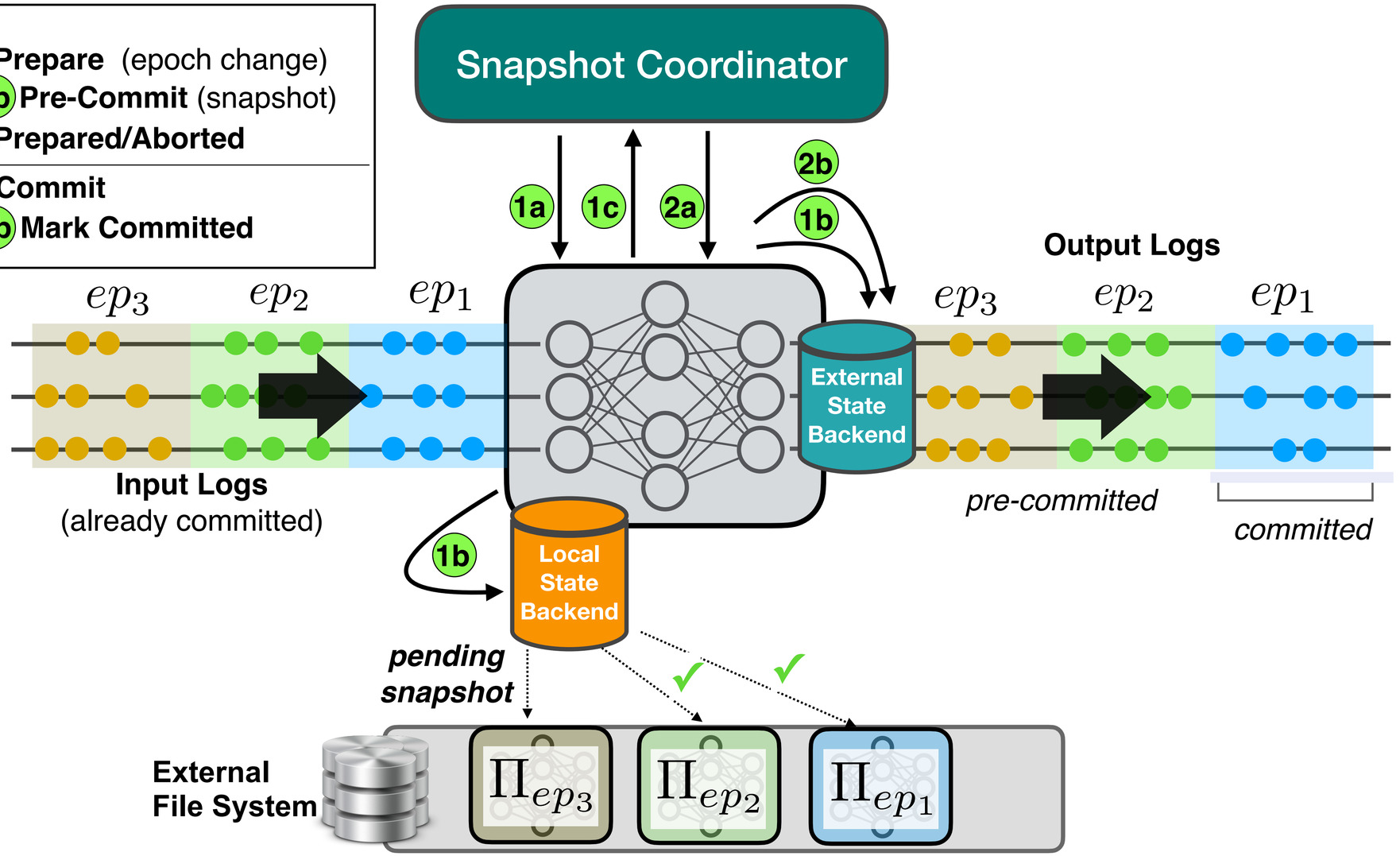
Epoch Snapshotting Algorithm



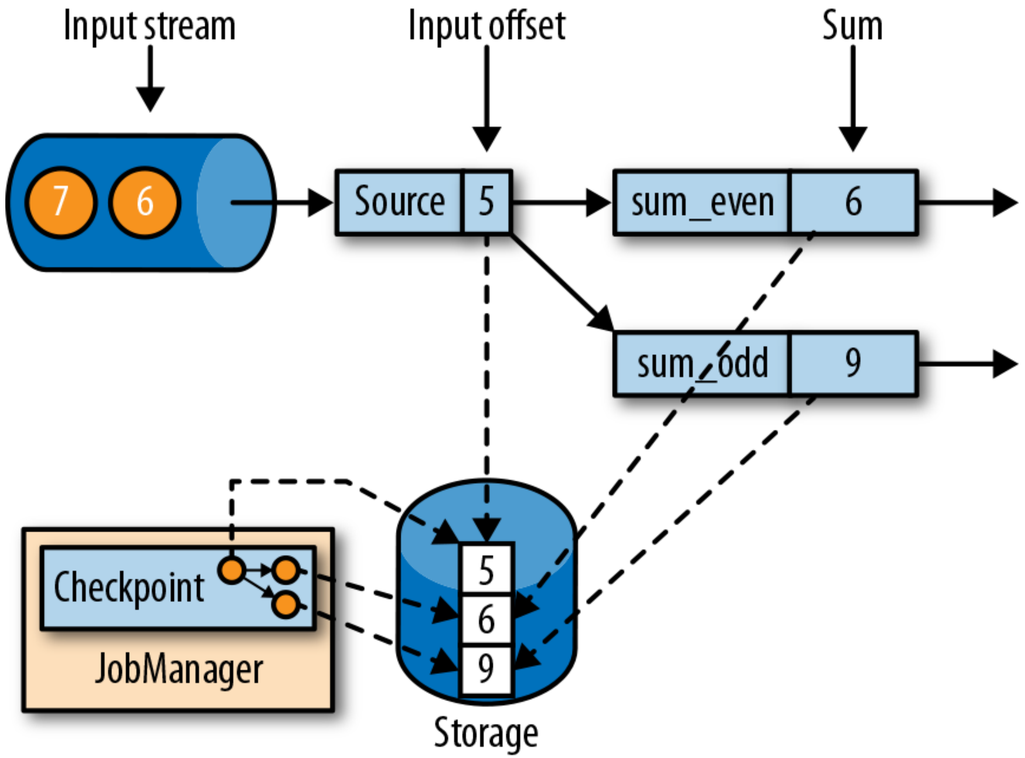
The Epoch Commit Protocol

1a **Prepare** (epoch change)
1b **Pre-Commit** (snapshot)
1c **Prepared/Aborted**

2a **Commit**
2b **Mark Committed**



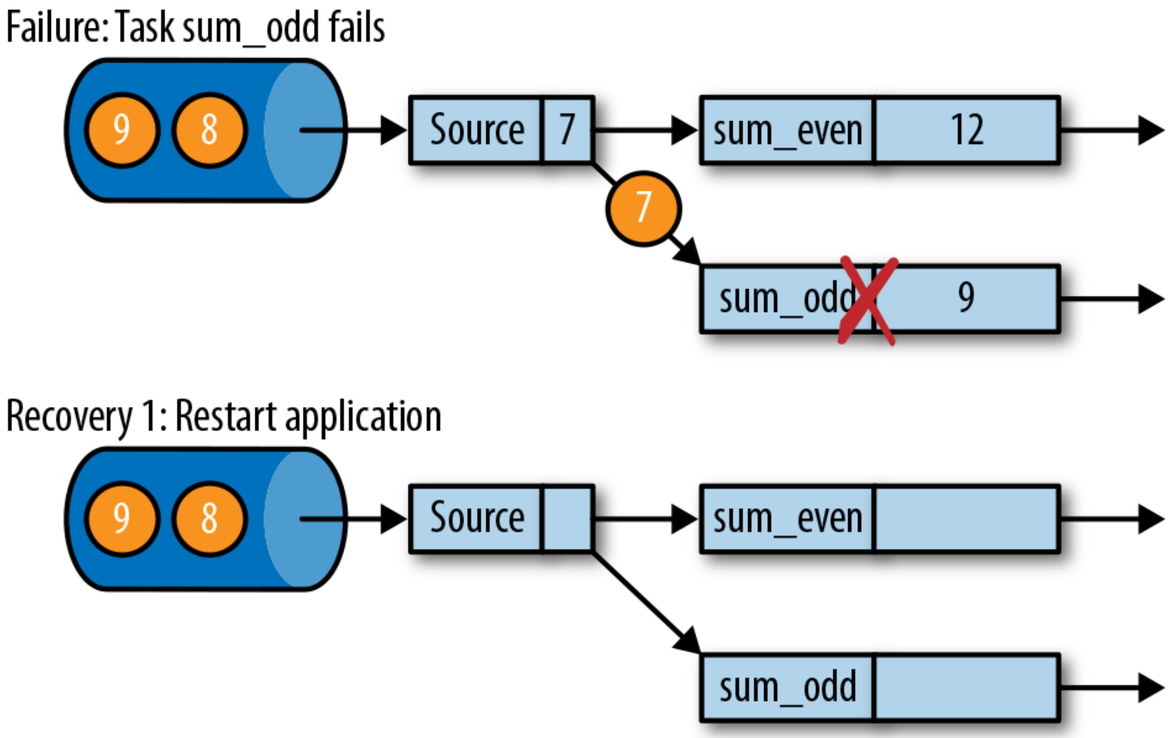
Asynchronous checkpoints in Apache Flink



- A source of increasing numbers partitioned into a stream of even and odd numbers
- Two sum operators maintaining the running sums of even and odd numbers
- The snapshot contains the source offset 5 and the sums 6 and 9

Recovery process

1. Stop and restart the application. All operators have empty state.

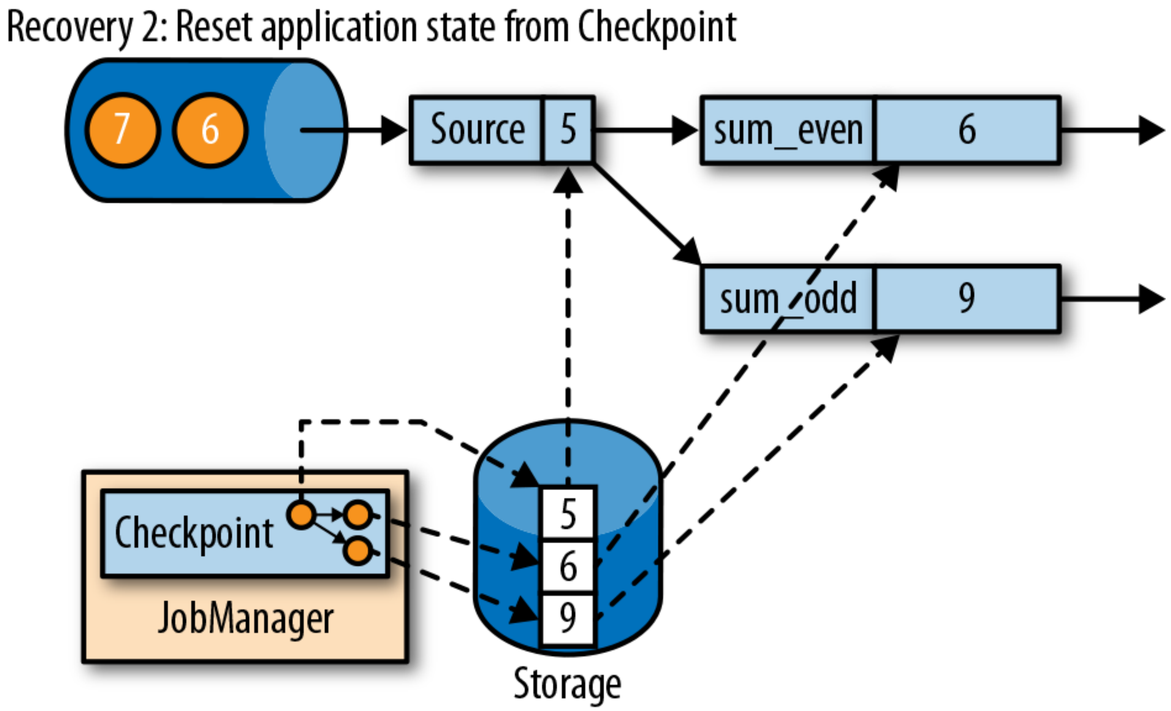


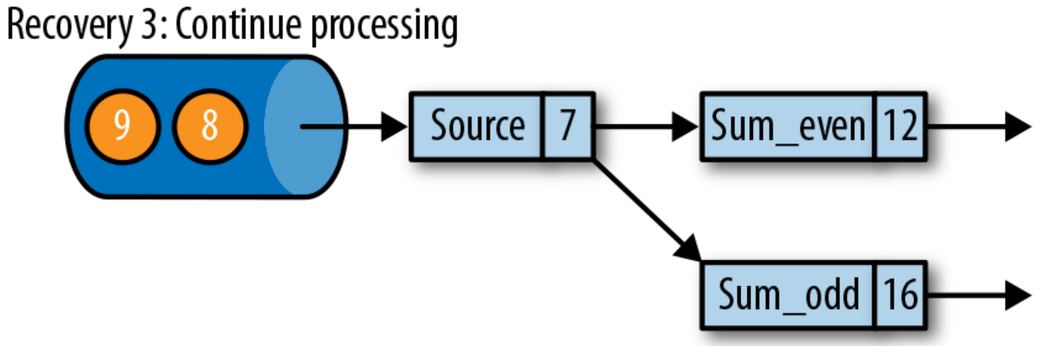
Recovery process

1. Stop and restart the application. All operators have empty state.

2. Initialize stateful operators from the latest checkpoint.

3. Resume processing.





Re-settable sources

- All input streams are reset to the position up to which they were consumed when the checkpoint was taken.
- Event logs like Apache Kafka can provide records from a previous offset of the stream.

Re-settable sources

- All input streams are reset to the position up to which they were consumed when the checkpoint was taken.
- Event logs like Apache Kafka can provide records from a previous offset of the stream.

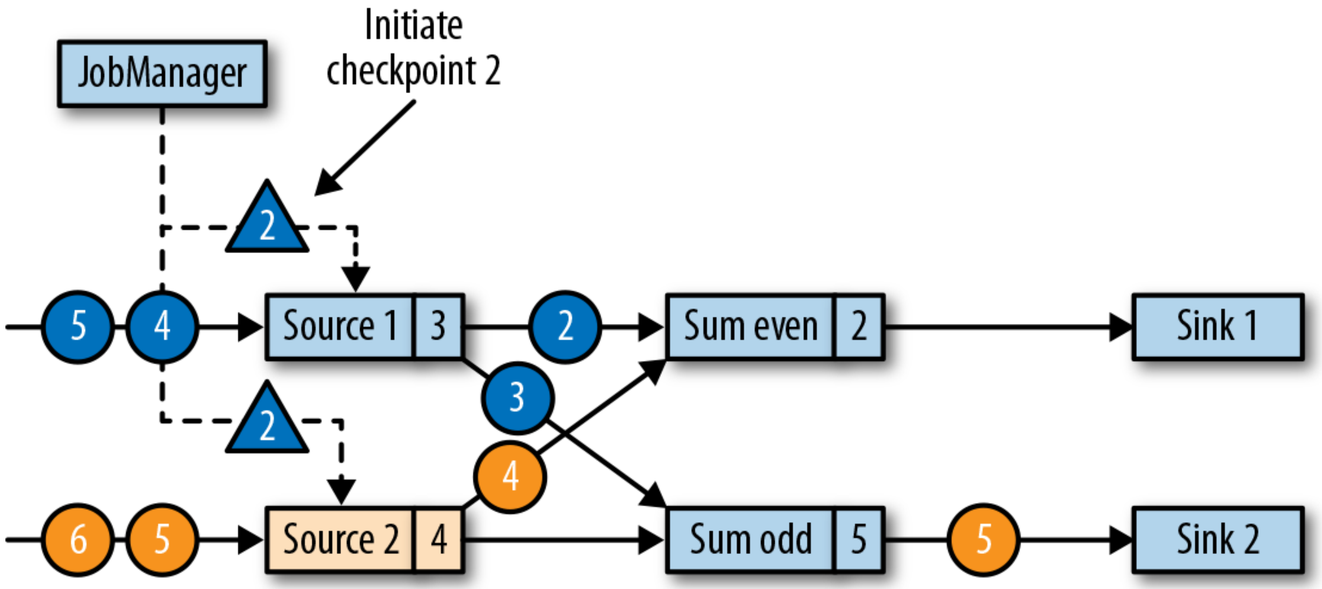
What if the input stream comes from a socket?

Re-settable sources

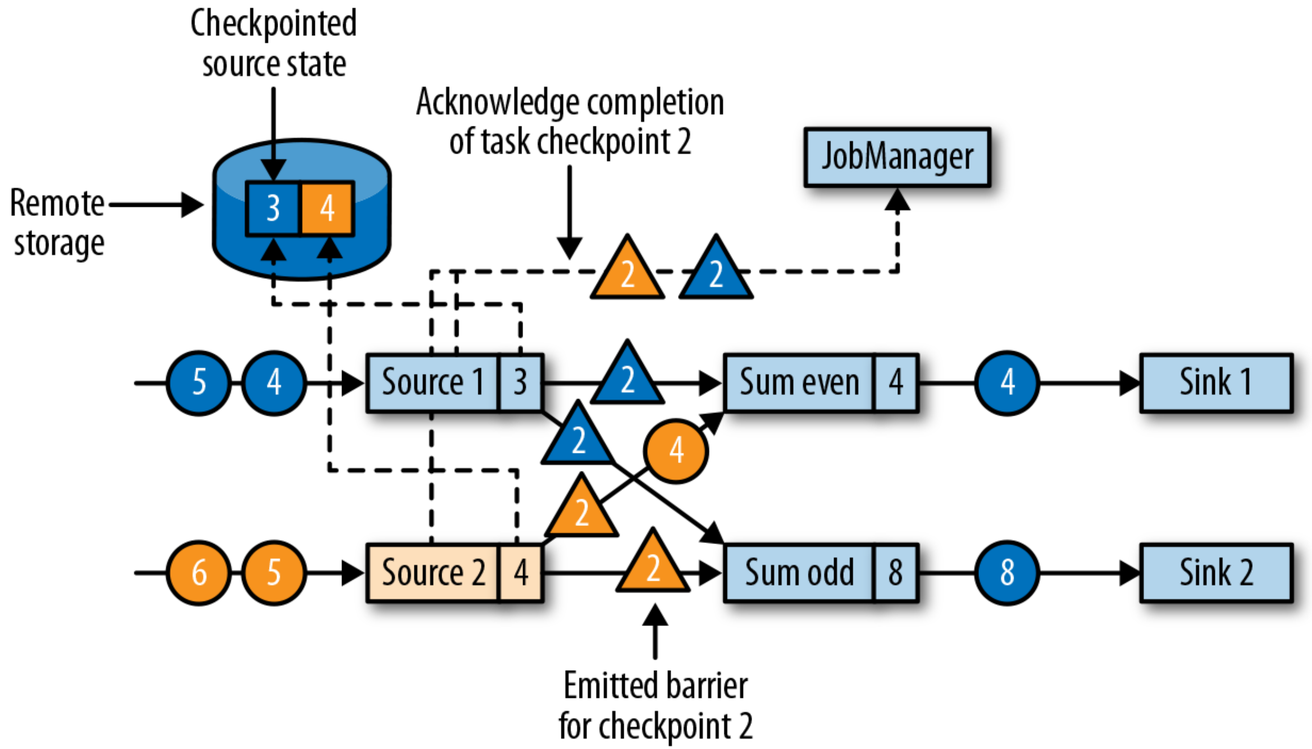
- All input streams are reset to the position up to which they were consumed when the checkpoint was taken.
- Event logs like Apache Kafka can provide records from a previous offset of the stream.

What if the input stream comes from a socket?

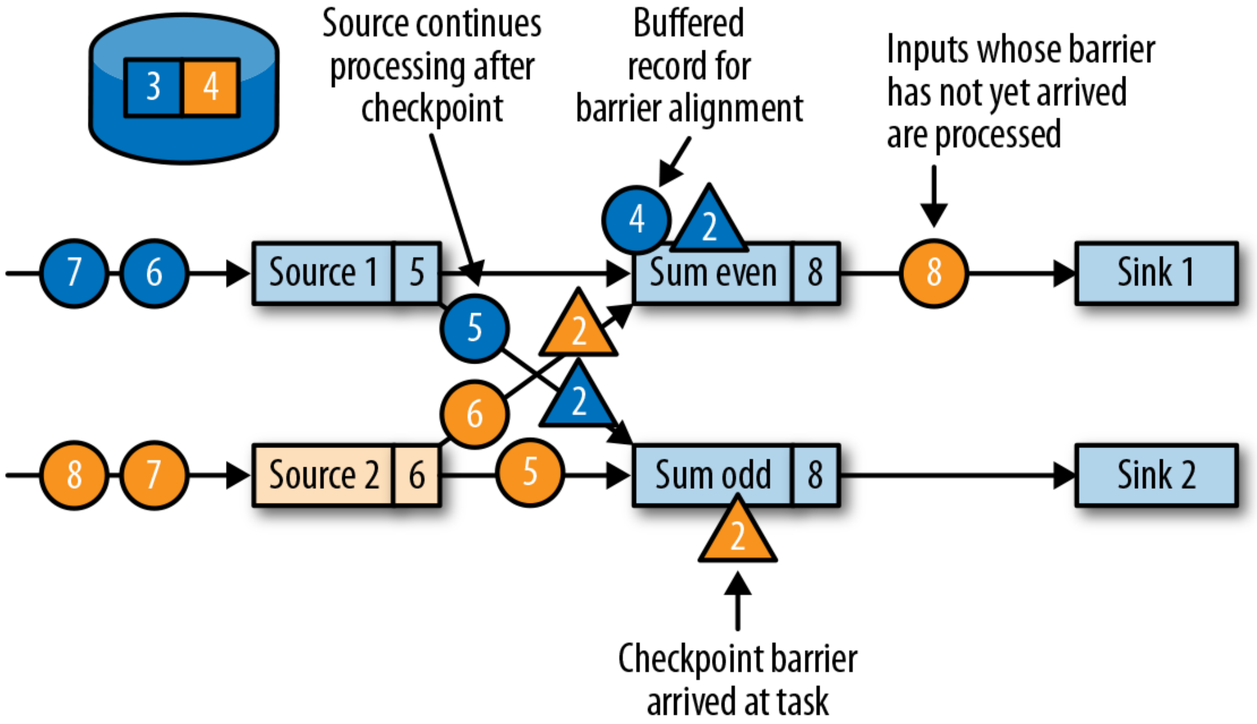
Exactly-once state consistency (in Apache Flink) can be achieved only if all streaming sources are re-settable



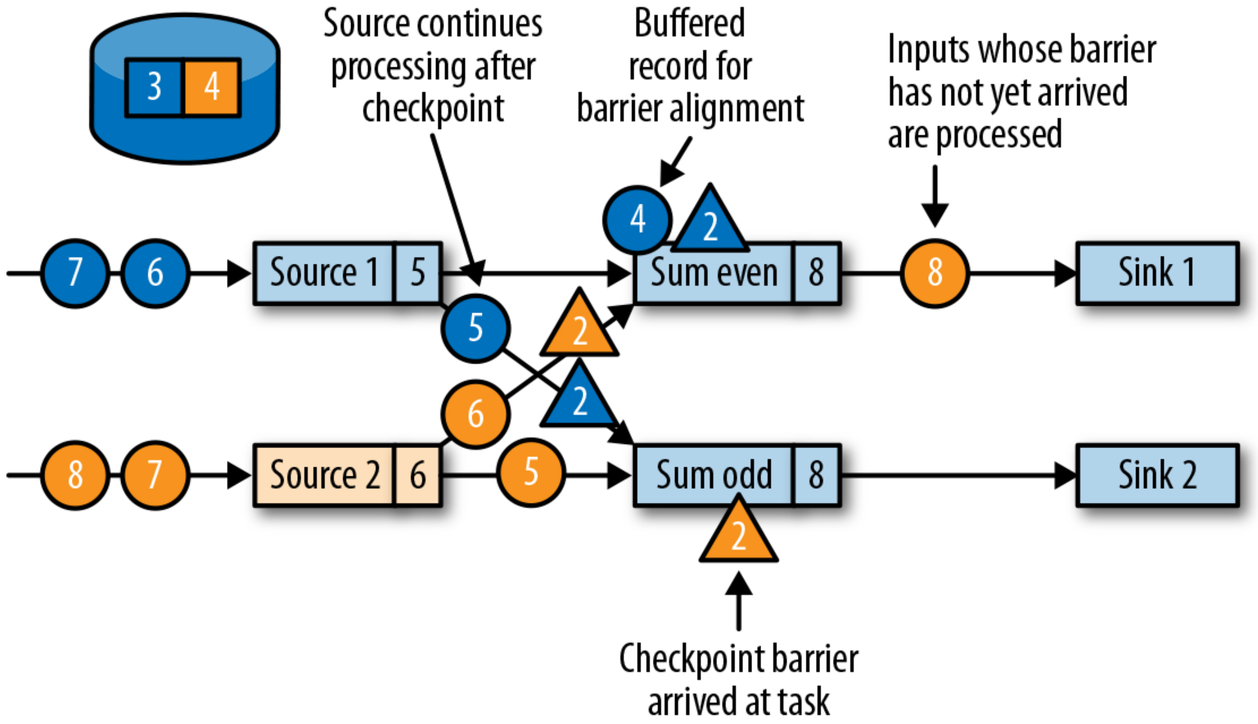
- Flink checkpoints are initiated by the JobManager and have a unique Checkpoint ID.
- The checkpoint barrier flows from the sources to the sink of the dataflow.



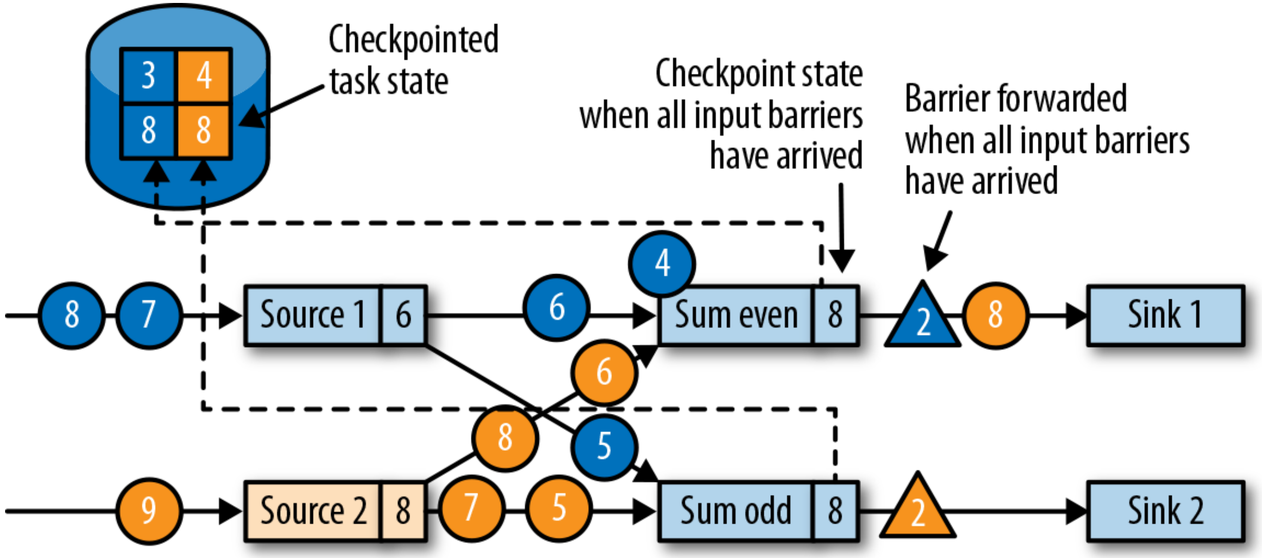
- When a source task receives a checkpoint barrier, it pauses emitting records, triggers a checkpoint of its local state at the state backend, and broadcasts barriers to all outgoing stream partitions.
- The state backend notifies the task once its state checkpoint is complete.
- The task acknowledges the checkpoint to the JobManager. After all barriers are sent out, the source continues its regular operations.

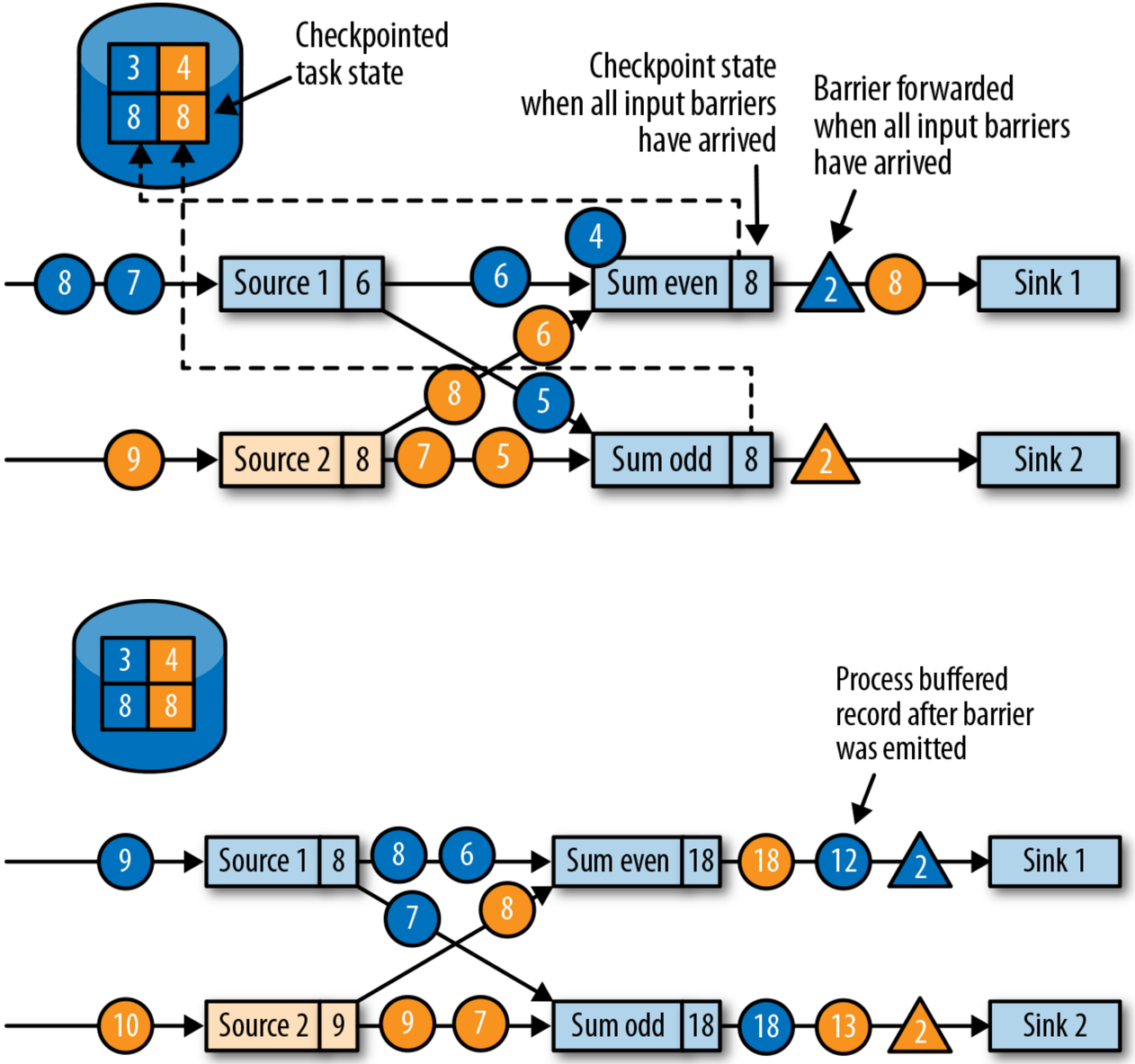


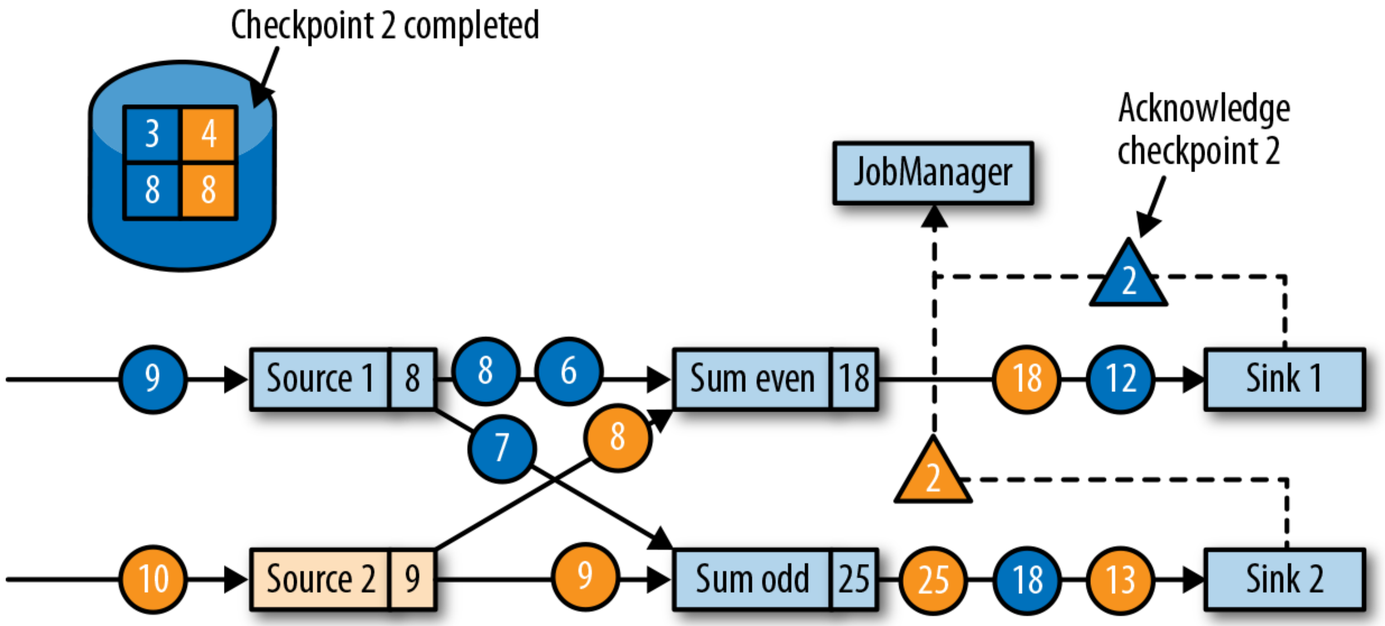
- When a task receives a checkpoint barrier, it waits for the arrival of barriers from all its input partitions for the checkpoint.
- In the meantime, it can process records from stream partitions that did not provide a barrier yet.
- Records that arrive on partitions that forwarded a barrier already cannot be processed and are buffered. This process is called **barrier alignment**.



How would state guarantees change if we skipped barrier alignment?


Checkpointed
task state
3
4
8
8
Checkpoint state
when all input barriers
have arrived
Barrier forwarded
when all input barriers
have arrived
8
7
Source 1
6
6
4
Sum even
8
2
8
Sink 1
6
8
5
9
Source 2
8
7
5
Sum odd
8
2
Sink 2


Checkpointed task state
Checkpoint state when all input barriers have arrived
Barrier forwarded when all input barriers have arrived
Source 1
Source 2
Sum even
Sum odd
Sink 1
Sink 2
Process buffered record after barrier was emitted



- When a sink task receives a barrier, it performs a barrier alignment, checkpoints its own state, and sends an acknowledgement to the JobManager.
- The checkpoint is completed once the JobManager has received acknowledgements from all tasks.

Performance implications

How may checkpointing affect application performance?

Performance implications

How may checkpointing affect application performance?

How often to checkpoint?

Performance implications

How may checkpointing affect application performance?

How often to checkpoint?

Do we need to checkpoint the complete application state in every checkpoint?

Performance implications

How may checkpointing affect application performance?

How often to checkpoint?

Do we need to checkpoint the complete application state in every checkpoint?

- RocksDB supports both **asynchronous** and **incremental** checkpoints:
  - take a local snapshot and use a background thread to copy the state to remote storage
  - compute state deltas to reduce data transfer

End-to-end exactly once

- Flink's checkpointing and recovery mechanism only resets the internal state of a streaming application
- Some result records might be emitted multiple times to downstream systems

End-to-end exactly once

- Flink's checkpointing and recovery mechanism only resets the internal state of a streaming application
- Some result records might be emitted multiple times to downstream systems

How can we ensure exactly-once output?

Enabling and configuring checkpoints

```
val env = StreamExecutionEnvironment.getExecutionEnvironment

// set checkpointing interval to 10 seconds
env.enableCheckpointing(10000L)
```

```
// get the CheckpointConfig from the StreamExecutionEnvironment
val cpConfig: CheckpointConfig = env.getCheckpointConfig

// set mode to at-least-once
cpConfig.setCheckpointingMode(CheckpointingMode.AT_LEAST_ONCE);

// make sure we process at least 30s without checkpointing
cpConfig.setMinPauseBetweenCheckpoints(30000);

// allow three checkpoints to be in progress at the same time
cpConfig.setMaxConcurrentCheckpoints(3);

// checkpoints have to complete within five minutes, or are discarded
cpConfig.setCheckpointTimeout(300000);

// do not fail the job on a checkpointing error
cpConfig.setFailOnCheckpointingErrors(false);
```

Lecture references

- Paris Carbone et al. **State management in Apache Flink®: consistent stateful distributed stream processing**. (PVLDB 2017).
- Fabian Hueske, and Vasiliki Kalavri. **Stream Processing with Apache Flink**. (O'Reilly Media '19).
- A video lecture on global snapshots: https://www.coursera.org/lecture/cloud-computing/1-2-global-snapshot-algorithm-hndGi